Efficient Direct-Access Ranked Retrieval

Mohsen Dehghankar
University of Illinois Chicago
mdehgh2@uic.edu

Raghav Mittal
University of Texas at Arlington
rxm0006@mavs.uta.edu

Suraj Shetiya
IIT Bombay
surajs@cse.iitb.ac.in

Abolfazl Asudeh
University of Illinois Chicago
asudeh@uic.edu

Gautam Das
University of Texas at Arlington
gdas@uta.edu

ABSTRACT

We study the problem of *Direct-Access Ranked Retrieval* (DAR) for interactive data tooling, where evolving data exploration practices, combined with large-scale and high-dimensional datasets, create new challenges. DAR concerns the problem of enabling efficient *access to arbitrary rank positions* according to a ranking function, without enumerating all preceding tuples.

To address this need, we formalize the DAR problem and propose a theoretically efficient algorithm based on geometric arrangements, achieving logarithmic query time. However, this method suffers from exponential space complexity in high dimensions.

Therefore, we develop a second class of algorithms based on ε-sampling, which consume a linear space. Since exactly locating the tuple at a specific rank is challenging due to its connection to the range counting problem, we introduce a relaxed variant called *Conformal Set Ranked Retrieval* (CSR), which returns a small subset guaranteed to contain the target tuple.

To solve the CSR problem efficiently, we define an intermediate problem, *Stripe Range Retrieval* (SRR), and design a hierarchical sampling data structure tailored for narrow-range queries. Our method achieves practical scalability in both data size and dimensionality.

We prove near-optimal bounds on the efficiency of our algorithms and validate their performance through extensive experiments on real and synthetic datasets, demonstrating scalability to millions of tuples and hundreds of dimensions.

1 INTRODUCTION

While recent advances in data tooling have significantly transformed the landscape of data science, the challenges of the AI era have posed new obstacles in data exploration and visual analytics [6]. Modern human-centered tools, such as spreadsheets, data science notebooks, visual analytics platforms, and interactive data science libraries, empower users with intuitive interfaces for seamless data exploration and manipulation [15, 43, 46, 52]. However, the exponential growth of data sizes due to the widespread adoption of big data technologies has placed unprecedented demands on the efficiency and responsiveness of these tools. As datasets scale to massive volumes, ensuring efficient and interactive user experiences becomes a critical yet challenging task.

A common practice in interactive data exploration is *ranked retrieval*, where data is ordered according to a user-defined or algorithmically generated ranking function, while the user may be interested in accessing *specific rank positions* or *a window* of ranked tuples. To further clarify this, let us consider the following example.

Example 1. *In spreadsheet applications [34, 44], data scientists frequently create new attributes as weighted sums of existing ones. Subsequently, they may* "jump" *to tuples at specific rank positions without explicitly enumerating preceding entries.*

Similar examples can be made for other applications. For instance, in Jupyter notebooks [32], analysts may need to directly retrieve tuples at particular ranks, effectively *skipping* over earlier-ranked items. Furthermore, recommendation systems frequently employ ranked retrieval for paged access, allowing users to directly navigate to specific result pages determined by complex ranking criteria.

In all these scenarios, the ability to perform *efficient direct-access ranked retrieval*, i.e., to quickly access the tuple at an arbitrary rank position i, without the overhead of enumerating the entire dataset up to that position, is crucial for maintaining the *interactivity* and *responsiveness* expected by users. Particularly, the large number of attributes in datasets within domains such as AI creates a *high-dimensional space of possible ranking functions*, potentially encompassing up to tens or hundreds of dimensions, particularly when ranking functions are machine-generated.

Traditionally, the interest of the data management community in ranked retrieval has been on *top-k* query processing, with the objective of finding the *top-ranked* tuples [30]. Such approaches, however, are tailored for a small value of k, and are ineffective for direct-access queries, with the goal of efficiently retrieving the tuple at an arbitrary ranking position i, where i can be in $O(n)$. Direct access to conjunctive queries has recently been studied to directly access a specific position in the join result, without fully materializing the join [9, 23, 49]. These works, however, consider an *prespecified ordering* of the tuples (e.g., lexicographic order on an attribute), and focus on skipping the join operation. As a result, such approaches are not suitable for ranked-retrieval settings where the ranking function is part of the input query.

Therefore, in this paper, we formalize and study the *direct-access ranked retrieval* (DAR) problem. We also introduce the notion of *conformal set* for ranked retrieval as a small set that contains the tuple at the queried rank i.

Next, focusing on linear ranking functions, we propose efficient algorithms for each of the proposed problems. Specifically, our first algorithm uses the computational geometric concepts of duality, arrangements, and levels of arrangements for answering the DAR problem. This algorithm, however, suffers from the curse of dimensionality, and its space complexity increases exponentially with the number of attributes.

Consequently, we propose several algorithms with a linear space complexity, even for a large number of attributes. Specifically, starting from $d = 2$, we propose our algorithm based on the concept of

ε-samples, and extend it to high-dimensional settings by combining it with the stripe-range searching. This approach offers an efficient solution for finding a conformal set for an arbitrary rank position. Finally, combining our approach with range counting, we extend it for the exact version (DAR problem).

Efficiently finding the tuples within a *narrow* stripe range is fundamental in the development of our ε-sampling based solution. We formulate this as the problem of *stripe range* retrieval (SRR), with a focus on developing efficient algorithms for the challenging narrow ranges that only contain a small number of tuples.

Inspired by the Hierarchical Navigable Small World graphs for approximate nearest neighbor problems [19, 35], our practical and scalable algorithm constructs a hierarchical sampling structure for stripe range searching, maintaining the neighborhood regions of points in the hierarchy using smallest enclosing balls.

Theoretically, we show the near-optimal efficiency of our algorithms up to a logarithmic factor. Besides, we conduct extensive experiments on real-world and synthetic datasets to evaluate the performance of our algorithms in practice and compare them against the existing baselines. Our experiments confirm the *efficiency and scalability* of our proposed algorithms. In particular, the ε-sampling algorithm, when combined with the hierarchical structure, achieves the best scalability with respect to both dimension and dataset size in terms of query time and index size, whereas the baseline methods are mostly affected by the curse of dimensionality.

Summary of Contributions.

- (Section 2) We introduce and formalize the problems of (i) direct-access ranked retrieval (DAR), (ii) conformal set ranked retrieval (CSR), and (iii) stripe range retrieval (SRR).
- (Section 4) We provide a lower bound on the DAR problem as well as proving the near-optimality of our algorithms up to a logarithmic factor (discussed in Appendix A in this Technical Report).
- (Section 5) We propose an algorithm based on the geometric concepts of duality and the arrangement of hyperplanes that offers a logarithmic query time, suitable for low-dimensional settings, especially when $d = 2$.
- (Section 6) We propose several algorithms with linear space usage based on ε-sampling combined with range searching to offer efficient solutions for the CSR and DAR problems.
- (Section 6.3) We propose a practical solution for the stripe range retrieval problem (SRR), based on hierarchical random sampling and keeping track of the proximity of points. This solution also enables a practical algorithm for the CSR problem, scalable to large-scale settings, where we have to deal with narrow stripe ranges.
- (Section 7) We conduct comprehensive experiments to demonstrate the practical performance of our algorithms across different settings on synthetic and real-world scenarios.

In addition, the background concepts relevant to our algorithms are reviewed in Section 3, the related work is discussed in Section 8, and the final remarks are provided in Section 9.

2 FORMAL DEFINITIONS

Data Model. We consider a dataset $\mathcal{D} = \{p_1, p_2, \cdots, p_n\}$ consisting of n points in $\mathbb{R}^d$, where each point $p \in \mathcal{D}$ has d real-valued

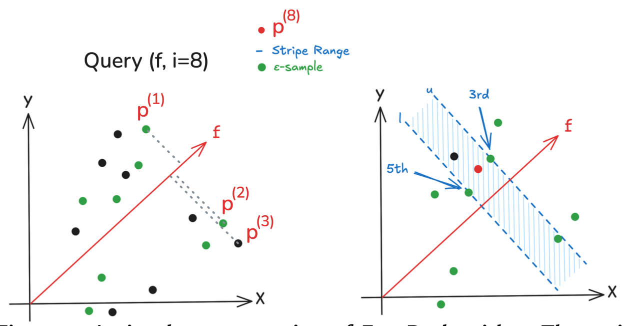

Figure 1: A visual representation of Eps2D algorithm. The main dataset contain 16 points and the query asks for $(i = 8)$-th point with the scoring function shown in red. (Left) First, we find the bounds by solving DAR on the ε-sample. (Right) Finding the ℓ and u values, we solve the SRR problem on the stripe range.

(scoring) attributes, aka. features, used for ranking.[1] A *scoring function* $\texttt{score} : \mathbb{R}^d \to \mathbb{R}$ assigns a real-valued score to each point in the dataset.[2] In this work, we focus on *linear scoring functions*, where the score is defined as a weighted sum of the features. Specifically, given a weight vector $f \in \mathbb{R}^d$, the score assigned to a point p is computed as the dot product[3]: $\texttt{score}_f(p) = f^\top p = \sum_{j=1}^{d} f[j]p[j]$. We may directly call f as the scoring function. Figure 1 (left) provides a toy example of a dataset $\mathcal{D}$ with $n = 16$ points and two attributes x and y. Let the red origin-anchored vector in the figure represent f. The perpendicular projection of each point p on the vector f shows $\texttt{score}_f(p)$.

2.1 Direct-Access Ranked Retrieval (DAR)

Rank. Given a scoring function f, we define a total order over the dataset $\mathcal{D}$ by sorting the points in descending order of their scores. For example, in Figure 1, the ordering of the projections on the vector of f shows their ranking. Let $\texttt{rank}_{\mathcal{D},f} : \mathcal{D} \to \mathbb{N}$ denote the function that assigns to each point in $\mathcal{D}$ its rank in this ordering (i.e., $\texttt{rank}_{\mathcal{D},f}(p) = i$ if p is the i-th highest-scoring point under $\texttt{score}_f$). When the dataset $\mathcal{D}$ is clear from the context, we write $\texttt{rank}_f$ for brevity. We denote by $p^{(i)}$ the i-th point (aka the i-th element) under $\texttt{score}_f$, that is,

$$p^{(i)} = \texttt{rank}_f^{-1}(i), \quad \forall\, 1 \le i \le n.$$

Our main problem in this paper is to efficiently find the point at a specific rank i.

Problem 1 (Direct-Access Ranked Retrieval (DAR)). *Given a dataset $\mathcal{D}$ available at preprocessing time, a query consists of a linear scoring function $f \in \mathbb{R}^d$ and an integer $i \in [n]$, find the point $p^{(i)}$, with rank i in $\mathcal{D}$ according to the scoring function $\texttt{score}_f$. i.e. return:*

$$p^{(i)} = \texttt{rank}_{\mathcal{D},f}^{-1}(i)$$

As proved in our technical report, finding $p^{(i)}$ in high-dimensional settings is challenging since its complexity is tied to the complexity of *half-space range counting* problem [1, 11].

[1]Besides scoring attributes, the dataset may contain other attributes such as non-ordinal descriptive attributes that are not used for ranking. In this paper, we use the terms attribute and feature interchangeably to refer to the scoring attributes.
[2]We use the terms scoring function and ranking function interchangeably.
[3]We assume that the scoring vector f is normalized, i.e., $\|f\| = 1$.

Therefore, inspired by the machine learning concept of conformal predictions [4, 45], we introduce a "relaxed" version of Problem 1, where instead of returning one point $p^{(i)}$, we return *a small set that contains* $p^{(i)}$ – referred as the *conformal set* for rank i.

Problem 2 (Conformal-Set Ranked Retrieval (CSR)). *Given a dataset $\mathcal{D}$ available at preprocessing time, a query consists of a linear scoring function $f \in \mathbb{R}^d$, an integer $i \in [n]$, and a small value $\kappa \ll n$, find a conformal set $C(i) \subset \mathcal{D}$, where $|C(i)| \leq \kappa$ and $\texttt{rank}^{-1}_{\mathcal{D},f}(i) \in C(i)$.*

2.2 Stripe Range Retrieval (SRR)

While addressing our main problems (Problem 1 and Problem 2), we also address a side problem that finds all points that fall in a stripe range. The solution to this problem helps us in developing efficient solutions for our main problems. Formally, we define a stripe range as follows.

Stripe Range. A *stripe range* $\mathcal{S}_{f,\ell,u}$ is defined by a scoring function $f \in \mathbb{R}^d$ and two real-valued boundaries $\ell, u \in \mathbb{R}$ such that $\ell \leq u$. It corresponds to the universe of all points $x \in \mathbb{R}^d$ whose scores based on f lies within the interval $[\ell, u]$, i.e.,

$$\mathcal{S}_{f,\ell,u} = \{x \in \mathbb{R}^d \mid \ell \leq \texttt{score}_f(x) \leq u\}.$$

Given a stripe range query, we aim to return the set of points in $\mathcal{D}$ that fall inside the given stripe range. For example, in Figure 1, the stripe range query is highlighted between the two blue lines, and the points within this query should be returned.

Problem 3 (Stripe Range Retrieval (SRR)). *Given a dataset $\mathcal{D}$ and a query consisting of a* stripe range $\mathcal{S}_{f,\ell,u}$, *return all points in $\mathcal{D}$ that lie within this range, i.e., $\mathcal{D}_o = \mathcal{S}_{f,\ell,u} \cap \mathcal{D}$.*

3 BACKGROUND

In this section, we summarize the key concepts and definitions used in the subsequent discussions. We assume a range space $(\mathcal{D}, \mathcal{R})$, where $\mathcal{D}$ is the dataset, and $\mathcal{R}$ is a collection of geometric ranges defined over $\mathcal{D}$, such as all possible balls, half-spaces, or other regions in $\mathbb{R}^d$.

VC-Dimension. The Vapnik-Chervonenkis (VC) dimension [50] is a fundamental combinatorial measure of complexity for a range space. Formally, the VC-dimension is the size of the largest subset $C \subseteq \mathcal{D}$ such that every subset of C can be realized as the intersection of C with some range in $\mathcal{R}$. A low VC-dimension implies that the range space has limited expressive power, which in turn allows for efficient sampling-based approximations (see [27] for more details).

ε-Samples. An ε-sample (or ε-approximation) of a range space $(\mathcal{D}, \mathcal{R})$ is a subset $\mathcal{N} \subseteq \mathcal{D}$ such that, for every range $R \in \mathcal{R}$, the proportion of points in R is approximately preserved in $\mathcal{N}$. More formally, $\mathcal{N}$ is an ε-sample if for all $R \in \mathcal{R}$,

$$\left| \frac{|R \cap \mathcal{D}|}{|\mathcal{D}|} - \frac{|R \cap \mathcal{N}|}{|\mathcal{N}|} \right| \leq \varepsilon.$$

For more details on ε-sampling, we refer the reader to [27].

We use the following foundational theorem on ε-samples as a key tool in deriving our theoretical guarantees:

Theorem 1 (ε-sample theorem [50]). *Let $(\mathcal{D}, \mathcal{R})$ be a range space of VC-dimension d. Then for any $\varepsilon, \varphi \in (0, 1)$, a random sample $S \subseteq \mathcal{D}$ of size $O\left(\frac{d}{\varepsilon^2} \log \frac{d}{\varepsilon} + \frac{d}{\varepsilon^2} \log \frac{d}{\varphi}\right)$ is an ε-sample of $(\mathcal{D}, \mathcal{R})$ with probability at least $1 - \varphi$.* □

ε-Sample Construction. In addition to random sampling, which provides a probabilistic method for constructing ε-samples, there also exist deterministic approaches based on discrepancy theory [12, 27]. These methods can achieve near-linear runtime while ensuring the desired approximation guarantees.

4 SOLUTION OVERVIEW

In this section, we provide a high-level overview of our proposed algorithms, with a summary of their theoretical guarantees.

The baseline approach for answering the DAR queries uses the extension of the classic divide and conquer (D&C) median finding algorithm, which, instead of the median, finds the element at position i [7, 16]. To do so, it first makes a linear pass over $\mathcal{D}$, and for each point p, computes $\texttt{score}_f(p)$. Then, using the D&C algorithm, it finds $\texttt{rank}^{-1}_f(i)$ in $O(n)$.

Instead, our algorithms, summarized in Table 1, provide more efficient solutions by preprocessing the data and constructing proper data structures. For a more detailed discussion on the related literature and baselines, refer to Section 8.

KthLevel. We begin by introducing an algorithm that leverages the construction of the level of the arrangement of hyperplanes to solve the exact DAR problem (see Section 5.1 for detailed definitions). This method offers highly efficient logarithmic query time, but it is not practical for high-dimensional settings: it suffers from the curse of dimensionality and has an exponential space complexity. Therefore, our subsequent algorithms based on ε-samples aim at maintaining a linear space complexity.

ε-sampling. In the simpler two-dimensional case, where the dataset $\mathcal{D}$ consists of points in the plane. During preprocessing, we compute an ε-sample of the dataset, denoted by $\mathcal{N}_\varepsilon$. Given a scoring function f and a target rank i, we approximately solve the DAR problem on the smaller set $\mathcal{N}_\varepsilon$, rather than the full dataset $\mathcal{D}$.

From this approximate solution, we derive a stripe range containing the true i-th ranked point. By solving an instance of range searching (SRR) we find this exact point. We refer to this algorithm as Eps2D. A visual representation of this approach is shown in Figure 1 (left and right).

For higher-dimensional settings, we follow a similar generalized strategy (EpsRange). However, unlike in 2D, standard range searching algorithms do not perform well in high-dimensional datasets. To overcome this, we propose a practical algorithm to solve the intermediate SRR problem with hierarchical sampling (EpsHier).

Lower Bounds and Optimality. A discussion on the lower bounds and optimality of our algorithms is provided in Appendix A in this Technical Report.

In the following sections, we describe the two classes of algorithms separately: (i) those based on constructing the *k-th level of arrangements*, and (ii) those based on *ε-sampling*.

[4]The Big-Oh notations in our guarantees hide constants that are polynomially dependent on the dimension d. Consistent with the computational geometry literature [1], our focus is on the exponential dependence on d.

Algorithm Name	Summary	Dimension	Space Complexity	Query Time	Problem
KthLevel	Following the levels of arrangement	$d \geq 2$	$O(n^d)$	$O(\log n)$	DAR
Eps2D	ε-sampling in 2D	$d = 2$	$O(n)$	$O(n^{2/3} \log n)$	DAR
EpsRange	ε-sampling + Range Searching	$d \geq 2$	$O(n)$	$O(\max\{n^{1-1/d}, \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\})$	CSR
				$O(\max\{n^{1-1/d}, n^{2/3} \log n\})$	DAR
EpsHier	ε-sampling + Hierarchical Sampling	$d \geq 2$	$O(n)$	Worst-case $O(n)$; Practically Efficient	CSR

Table 1: Overview of the proposed methods for the DAR problem.[4]

5 ALGORITHM BASED ON k-TH LEVEL OF ARRANGEMENT

In this section, we present an algorithm for solving the DAR problem using the computational geometric concepts of *duality* and *arrangement* of hyperplanes [14, 22]. At a high level, building the arrangement of dual hyperplanes of the tuples, our approach, named KthLevel, keeps track of different *levels of the arrangement*, which are used at the query time for efficient answering of DAR queries, i.e., finding the tuple at a given rank position i. This algorithm achieves a *logarithmic query time*, but at a cost of a *space complexity exponential to the number of attributes d*.

We begin by introducing key concepts from computational geometry, and then describe our algorithm in detail.

5.1 Preliminaries

5.1.1 Duality. We use the classical dual space transformation that maps points to hyperplanes and vice versa. This transformation is well-defined and preserves intersection relationships [14, 22]. Formally, the dual of a point $p \in \mathbb{R}^d$ is defined as:

$$\mathsf{dual}(p) := \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^\top p = 1 \right\}.$$

In other words, the transformation $\mathsf{dual}(\cdot)$ maps a point in the *primal space* to a hyperplane in the *dual space*. This hyperplane is represented by the equation:

$$h : \sum_{i=1}^{d} p[i] \cdot x_i = 1.$$

We refer to the original input domain as the *primal space*, and the space of dual hyperplanes as the *dual space*. Figure 2 provides a visual illustration of this transformation. The left figure shows the primal space, while the middle figure shows the dual hyperplanes in the dual space. The following key observation follows from this duality. A formal proof can be found in [22].

Lemma 2. *Let f be a scoring function (i.e., a direction vector), and let $p^{(1)}, \ldots, p^{(n)}$ be the points sorted in descending order according to their score under f, so that $f^\top p^{(1)} > f^\top p^{(2)} > \cdots$. Then, for any $a < b$, the intersection point of $\mathsf{dual}(p^{(b)})$ with the ray in direction f lies closer to the origin than that of $\mathsf{dual}(p^{(a)})$.* □

For instance, in Figure 2, the point B has the highest score under f_1, and thus its dual $\mathsf{dual}(B)$ intersects the ray f_1 in the dual space earlier than any other point's dual.

5.1.2 k-th Level of Arrangement. Consider the set of dual hyperplanes $\mathcal{H} = \{\mathsf{dual}(p) \mid p \in \mathcal{D}\}$. $\mathcal{H}$ defines a dissection of $\mathbb{R}^d$ into connected convex cells, called the *arrangement* of the hyperplanes [22].

To simplify the explanations, let $d = 2$. Sweep a ray f counterclockwise from the x-axis to the y-axis. At each orientation of f, the ray intersects all dual hyperplanes (dual lines) in $\mathcal{H}$. These intersection points can be sorted by their distance from the origin. The *k-th level* is the locus of points that lie on the k-th closest dual line intersected by f, as the direction of f varies continuously [22].

In general d-dimensional space, consider a ray f originating at the origin and sweeping over all directions on the unit sphere $\mathbb{S}^{d-1}$. For each direction f, the ray intersects the hyperplanes in $\mathcal{H}$ in up to n distinct points, which can again be ordered by increasing distance from the origin. The *k-th level* is the set of all such k-th intersection points across all directions $f \in \mathbb{S}^{d-1}$.

Figure 2 illustrates the first and second levels of the arrangement in $\mathbb{R}^2$, shown in orange and green, respectively.

5.2 KthLevel Algorithm

In this subsection, we formally present our algorithm, KthLevel. We begin by describing the *preprocessing* phase, followed by a discussion of the *query phase*, and conclude with a theoretical analysis of the algorithm's performance. The overall structure of the algorithm applies uniformly across dimensions $d \geq 2$; however, for the purpose of analysis, we distinguish between the planar case ($d = 2$) and the general higher-dimensional case ($d > 2$).

5.2.1 Preprocessing. During the preprocessing phase, we construct the arrangement of the dual hyperplanes induced by the point set $\mathcal{D}$, and the levels of the arrangement for all $1 \leq k \leq n$. We assume access to an oracle $\mathcal{K}$ that, for a given i, returns an efficient data structure representing the i-th level of the arrangement [22]. By calling this oracle for each $i \in [n]$, we obtain the complete set of structures needed for query processing. Each such structure allows constant-time access to the hyperplanes intersecting on each face of the arrangement, as well as the points in the primal space corresponding to those hyperplanes [22].

5.2.2 Query Phase. In the query phase, we are given a scoring function f and a rank i, and the goal is to retrieve the point $p^{(i)}$, which is the i-th highest-ranked point with respect to f. By Lemma 2, the intersection of the i-th level of arrangement of dual hyper-planes with f corresponds with the desired point $p^{(i)}$. Algorithm 1 shows the pseudo-code of our process for finding this intersection.

Consider the 3rd level of the arrangement highlighted (in orange) in the Figure 2 (right figure). In 2D, the k-th level is a non-convex chain of line segments, each belonging to a dual line. Now, let the query function be f_2, while $i = 3$. Our objective is to find $p^{(3)} = \mathsf{rank}^{-1}_{\mathcal{D},f_2}(3)$. The line segment in the 3rd level that intersects

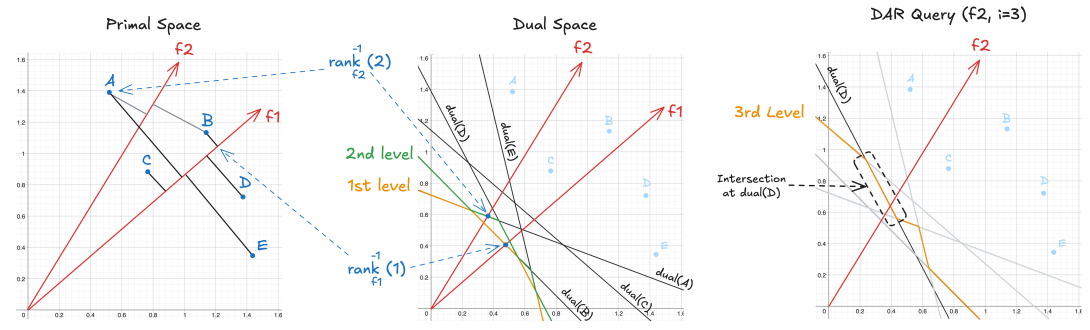

Figure 2: A 2D representation of dual space and the k-th level of arrangements (For simplicity, only the first quadrant is presented). The right figure shows an example of running KthLevel for $i = 3$. The result of this query is the point D.

Algorithm 1 KthLevel-Query(f, i)

```
Input: The scoring function $f$ and the order $i$
Output: The $i$-th highest scored point according to $f$: $p^{(i)}$
1: $Level_i \leftarrow$ preprocessed data structure        ▷ Get the $i$-th level
2: $face \leftarrow$ Intersection(f, Level$_i$)        ▷ Get the intersection
3: $h \leftarrow$ The hyperplane corresponding to $face$ in arrangement
4: $p^{(i)} \leftarrow \text{dual}^{-1}(h)$
5: return $p^{(i)}$
```

with the vector of f_2 is dual(D). Hence, the point $p^{(3)} = D$ in this example. Note that applying a binary search on the endpoints of the line segments of the 3rd level, one can find the intersection of any function f with any level i in a time logarithmic to the number of line segments.

From the above example, observe that the k-th level is a chain of line segments. Similarly, for a general $d \geq 2$, the k-th level is a chain of $(d-1)$-face.[5] Each function f, in such cases, is an origin-anchored vector that intersects one of the faces of each level $i, \forall i \in [n]$. Projecting the endpoints of these faces on the surface of the unit d-ball (the d-dimensional hypersphere with a unit radius), forms an arrangement in the $d-1$-dimensional space. For example, when $d = 3$, each k-th level is a chain of plane segments, and their projection onto the unit ball (centered at the origin) is an arrangement of lines.

Now, consider the intersection point of the query function f with the unit d-ball. Finding the face segment of a level i with a function i is equivalent to finding the cell of the arrangement on the surface of the ball that contains the intersection point of f. This problem is known as *point location* on the arrangement of hyperplanes [38]. Line 2 of Algorithm 1 uses a state-of-art point-location algorithm for finding the intersection of f with the i-th level.

5.3 Analysis of KthLevel Algorithm

5.3.1 Preprocessing. In 2D, the complexity of the k-th level in an arrangement of n lines in the plane is $O(nk^{1/3})$ [20]. Hence, considering an arbitrary level $i = O(n)$, the complexity of the i-th level is $O(n^{4/3})$. The total preprocessing time to construct all levels of the arrangement is $O(n^2)$, where each individual level can be built in $O(n \log n + nk)$ time [24]. In 3D, the complexity of the k-th level is $O(nk^{3/2})$ [47]. As a result, the complexity of each level $i \in [n]$ is bounded by $O(n^{5/2})$. The complexity of the arrangement in the general d-dimensional setting is $O(n^d)$, and the total time required to construct it is also $O(n^d)$ [2, 22].

5.3.2 Query Time. During the query time, Algorithm 1 finds the intersection of the i-th level with the vector of the function f. Given that the total number of faces in each level $i \in [n]$ is bounded by $O(n^d)$, identifying this intersection using a state-of-the-art algorithm for point location on the arrangement of hyperplanes is in $O(\log n)$, ignoring the constant factors that depend on d [38].

6 ε-SAMPLING APPROACH

In theory, the solution based on constructing the arrangement of dual hyperplanes provides a logarithmic time for answering the DAR queries. It, however, is not practical especially when the number of ranking attributes is not small (≥ 3), since the size of the arrangement (hence, the space complexity) increases exponentially with d. Therefore, in this section, we propose practical solutions that keep the space complexity linear, i.e., $O(n)$. Our algorithms are based on the computational geometric concept of ε-samples [27] (also introduced in Section 3).

In the following, we start by discussing the two-dimensional Eps2D algorithm, then, we will generalize our solution to the higher dimensions, and propose practical methods to solve the intermediate SRR problem (Problem 3).

6.1 Eps2D: ε-sampling in 2D

In this section, we discuss our solution based on ε-samples in 2D, called Eps2D.

Preprocessing. During the preprocessing phase, the algorithm computes an ε-sample of the dataset $\mathcal{D}$ by considering the *stripe ranges*, denoted by $\mathcal{N}_\varepsilon$, for a given parameter ε (see Section 3 for background).

Query Answering. Let $m = |\mathcal{N}_\varepsilon|$ denote the size of the ε-sample build during the preprocessing time. Note that $m \ll n$.[6]

[5]A d'-face of an arrangement is a d'-dimensional convex cell, formed by the intersection of $d - d'$ hyperplanes. For example, 0-faces are vertices (points), 1-faces are edges (segments or lines), and so on.

[6]The VC-dimension of stripe ranges is $d + 1$. Hence, it is 3 for $d = 2$.

Step i: Thresholding. During the *query phase*, given a query pair (f, i), we begin by solving the DAR problem on the ε-sample $\mathcal{N}_\varepsilon$. Specifically, we identify the i_ℓ-th and i_u-th ranked points in $\mathcal{N}_\varepsilon$ with respect to the scoring function f, where:

$$i_\ell = \left\lfloor m\left(\frac{i}{n} - \varepsilon\right)\right\rfloor, \quad i_u = \left\lceil m\left(\frac{i}{n} + \varepsilon\right)\right\rceil, \tag{1}$$

and $m = |\mathcal{N}_\varepsilon|$. This step can be done by sorting the points in $\mathcal{N}_\varepsilon$ according to $\texttt{score}_f$ or it can be done with linear time in m by using the baseline median finding algorithms, like median of medians.

These two points serve as score-based thresholds that are guaranteed to bound the score of the i-th ranked point in the full dataset $\mathcal{D}$, as stated in the following lemma.

Lemma 3. *Let $\mathcal{N}_\varepsilon$ be an ε-sample of the dataset $\mathcal{D}$ for some $\varepsilon > 0$. Then, for any linear scoring function f, the score of the i-th ranked point $p^{(i)}$ in $\mathcal{D}$ is bounded as*

$$\texttt{score}_f(q_\ell) \le \texttt{score}_f(p^{(i)}) \le \texttt{score}_f(q_u),$$

where q_ℓ and q_u are the i_ℓ-th and i_u-th ranked points in $\mathcal{N}_\varepsilon$ under f, i.e.,

$$q_j = \texttt{rank}^{-1}_{\mathcal{N}_\varepsilon, f}(i_j), \quad \textit{for } j \in \{\ell, u\}.$$

□

Proof. Let $\mathcal{H}$ be the half-space defined by:

$$\mathcal{H} = \{x \in \mathbb{R}^d \mid f^\top x \ge f^\top p^{(i)}\}.$$

This half-space is perpendicular to the direction vector f and contains exactly the top i points of $\mathcal{D}$ ranked by f, including $p^{(i)}$. In other words,

$$|\mathcal{H} \cap \mathcal{D}| = i.$$

Since $\mathcal{N}_\varepsilon$ is an ε-sample of $\mathcal{D}$ for half-space (and slab) ranges, it preserves the proportion of points in any half-space up to additive error ε. Therefore,

$$\left|\frac{|\mathcal{H} \cap \mathcal{N}_\varepsilon|}{|\mathcal{N}_\varepsilon|} - \frac{|\mathcal{H} \cap \mathcal{D}|}{|\mathcal{D}|}\right| \le \varepsilon.$$

Substituting $|\mathcal{H} \cap \mathcal{D}| = i$, and denoting $n = |\mathcal{D}|$ and $m = |\mathcal{N}_\varepsilon|$, we obtain:

$$\left|\frac{|\mathcal{H} \cap \mathcal{N}_\varepsilon|}{m} - \frac{i}{n}\right| \le \varepsilon,$$

which implies:

$$m\left(\frac{i}{n} - \varepsilon\right) \le |\mathcal{H} \cap \mathcal{N}_\varepsilon| \le m\left(\frac{i}{n} + \varepsilon\right).$$

Rounding to integers, define:

$$i_\ell = \left\lfloor m\left(\frac{i}{n} - \varepsilon\right)\right\rfloor, \quad i_u = \left\lceil m\left(\frac{i}{n} + \varepsilon\right)\right\rceil.$$

These bounds indicate that the number of points in $\mathcal{N}_\varepsilon$ with score at least $\texttt{score}_f(p^{(i)})$ lies between i_ℓ and i_u. Equivalently, the score $\texttt{score}_f(p^{(i)})$ lies between the scores of the i_u-th and i_ℓ-th ranked points in $\mathcal{N}_\varepsilon$. We conclude that:

$$\texttt{score}_f(q_\ell) \le \texttt{score}_f(p^{(i)}) \le \texttt{score}_f(q_u),$$

as desired. □

Algorithm 2 Eps2D-Query($\mathcal{N}_\varepsilon, \mathcal{D}, f, i$)

```
Input: Precomputed ε-sample $\mathcal{N}_\varepsilon$ of size $m$, original dataset $\mathcal{D}$,
    scoring vector $f$, target rank $i$
Output: The exact $k$-th ranked point $p^{(i)}$ in $\mathcal{D}$ under $\texttt{score}_f$
1: Calculate $i_\ell$ and $i_u$                                   ▷ equation 1
2: $q_\ell \leftarrow \texttt{rank}^{-1}_{\mathcal{N}_\varepsilon,f}(i_\ell)$,  $q_u \leftarrow \texttt{rank}^{-1}_{\mathcal{N}_\varepsilon,f}(i_u)$   ▷ Median of medians algorithm
3: $\ell \leftarrow \texttt{score}_f(q_\ell)$,  $u \leftarrow \texttt{score}_f(q_u)$          ▷ end of step i
4: $\mathcal{D}_o \leftarrow \mathcal{S}_{f,\ell,u} \cap \mathcal{D}$                     ▷ via SRS query; step ii
5: $|H_u| \leftarrow \left|\{p \in \mathcal{D} \mid f^\top p \ge u\}\right|$         ▷ via range counting; step iii
6: Sort $\mathcal{D}_o$ ascending by $\texttt{score}_f$
7: return the $(i - |H_u|)$-th point in the sorted list         ▷ step iv
```

Step ii: SRR. Next, we compute the score thresholds corresponding to the boundary points identified in the previous step:

$$\ell = \texttt{score}_f(q_\ell), \quad u = \texttt{score}_f(q_u). \tag{2}$$

Using these values, we define the stripe range $\mathcal{S}_{f,\ell,u}$. We then solve an instance of the SRR problem on the dataset $\mathcal{D}$ using this stripe range. The result is a subset of points within the score interval $[\ell, u]$, which we denote by $\mathcal{D}_o$. This set contains the i-th ranked point $p^{(i)}$, since $\mathcal{N}_\varepsilon$ is an ε-sample. Note that this set is a conformal set for rank i, hence providing a solution to the CSR problem (Problem 2).

Step iii: Range Counting. We define the half-space H_u as follows:

$$H_u = \{p \in \mathcal{D} \mid p^\top f \ge u\}, \tag{3}$$

which contains all points in $\mathcal{D}$ whose score under f is at least u. We then apply a half-space range counting algorithm in two dimensions to compute the number of points lying within H_u [36].

Step iv: Final Selection. In the final step, we sort the points in the candidate set $\mathcal{D}_o$ in descending order according to the scoring function $\texttt{score}_f$. Let $|H_u|$ denote the number of points in $\mathcal{D}$ whose score is greater than or equal to u, as computed in the previous step. We then return the $(i - |H_u|)$-th point in the sorted list as the exact answer to the DAR query.

A pseudo-code of the query phase of Eps2D is shown in Algorithm 2.

6.1.1 *Analysis.*

We analyze the time and space complexity of the Eps2D algorithm step-by-step.

Step i. Finding the i_ℓ-th and i_u-th point in $\mathcal{N}_\varepsilon$ of size m takes $O(m)$ with median finding algorithms.

Step ii. This step involves solving a simplex range searching query in 2D (with stripe ranges). Using the Matoušek's efficient range searching technique [36], this can be done in $O(\sqrt{n} + |\mathcal{D}_o|)$ time with $O(n)$ space.

Step iii. We perform a half-space range counting query. Again using Matoušek's partition tree [36], this takes $O(\sqrt{n})$ time and $O(n)$ space.

Step iv. Finally, we sort the set $\mathcal{D}_o$, which requires $O(|\mathcal{D}_o| \log |\mathcal{D}_o|)$ time.

Space Complexity. All operations use data structures with linear space requirements. Thus, the total space usage is linear in input.

Query Time Complexity. To analyze the runtime precisely, we bound the size of the result set $\mathcal{D}_o$ as follows:

Lemma 4. *The size of the result set $\mathcal{D}_o$ is $O(\varepsilon n)$.* □

Proof. The result set $\mathcal{D}_o$ consists of the points in the dataset $\mathcal{D}$ that lie between the i_ℓ-th and i_u-th ranked points in the sample $\mathcal{N}_\varepsilon$. Define the corresponding score interval as the stripe range:

$$S = \{x \in \mathbb{R}^d \mid f^\top q_\ell \le f^\top x \le f^\top q_u\}.$$

Then, $|\mathcal{D}_o| = |S \cap \mathcal{D}|$. Since $\mathcal{N}_\varepsilon$ is an ε-sample of $\mathcal{D}$ for stripe ranges, we have (with high probability):

$$\left| \frac{|S \cap \mathcal{N}_\varepsilon|}{|\mathcal{N}_\varepsilon|} - \frac{|\mathcal{D}_o|}{|\mathcal{D}|} \right| \le \varepsilon.$$

From Equation 1, we know that $|S \cap \mathcal{N}_\varepsilon| = |i_u - i_\ell| \le 2m\varepsilon + 2$. Hence:

$$\left| \frac{|S \cap \mathcal{N}_\varepsilon|}{m} - \frac{|\mathcal{D}_o|}{n} \right| \le \varepsilon$$
$$\implies \frac{|\mathcal{D}_o|}{n} \le \frac{2m\varepsilon + 2}{m} + \varepsilon$$
$$\implies |\mathcal{D}_o| = O(\varepsilon n).$$

□

Observation. In 2D, the size of the ε-sample satisfies (see Section 3):

$$m = O\left(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\right).$$

Total Runtime. Combining all steps, the total runtime is:

$$T(\text{Eps2D}) = O\left(m + \sqrt{n} + |\mathcal{D}_o| \log |\mathcal{D}_o|\right).$$

Using $|\mathcal{D}_o| = O(\varepsilon n)$ and choosing $\varepsilon = n^{-1/3}$ (which minimizes the overall runtime), we get:

$$T(\text{Eps2D}) = O\left(n^{2/3} \log n\right).$$

Theorem 5. *The Eps2D algorithm solves the exact DAR problem in 2D using linear space and $O(n^{2/3} \log n)$ query time.*

6.2 EpsRange: ε-sampling in Higher Dimensions

Next, we present the generalization of the Eps2D algorithm to higher dimensions ($d \ge 2$), referred to as the EpsRange algorithm. The preprocessing phase remains unchanged: we compute an ε-sample $\mathcal{N}_\varepsilon$ of the dataset $\mathcal{D}$, of size m, for a chosen parameter ε. Here, we know that $m \ll n$, because the VC-dimension of stripe ranges is $d + 1$ [27].

During query time, the algorithm proceeds similarly to Eps2D. In Step i, we find the i_ℓ-th and i_u-th points in $\mathcal{N}_\varepsilon$ according to the scoring function f. In Step ii, we solve a high-dimensional instance of the SRR problem. The resulting set $\mathcal{D}_o$ contains candidate points from $\mathcal{D}$ that lie between the scores of the i_ℓ-th and i_u-th ranked points in the sample. This set is a conformal set for Problem 2. To obtain the exact solution, we proceed to Step iii, where we perform a high-dimensional range counting query to determine the exact rank of each point in $\mathcal{D}_o$. Finally, in Step iv, we sort the candidates in $\mathcal{D}_o$ and return the point with exact rank i, completing the solution to the DAR selection problem.

6.2.1 Analysis.

The analysis is similar to Eps2D, however the time complexity of the algorithm depends on the specific range counting and simplex range searching we use at steps ii and iii.

Step i takes $O(m)$ for finding the thresholds in the ε-sample. Step ii, requires solving the SRR problem. Following the simplex range searching algorithms, we can solve this problem in $O(n^{1-1/d}+|\mathcal{D}_o|)$ with $O(n)$ space [37].

To report the exact response to DAR, which is $p^{(i)}$, we proceed to steps iii and iv. In step iii, we solve an instance of range counting in d-dimension which again takes $O(n^{1-1/d})$ [37] time and linear space. Finally, in step iv, we sort $\mathcal{D}_o$, which takes $O(|\mathcal{D}_o| \log |\mathcal{D}_o|)$.

Space Complexity. The space complexity of this algorithm is linear to input size, since all the structures used require linear space.

Time Complexity for the CSR problem. The total query time of the algorithm for finding the conformal set is:

$$T(\text{EpsRange}_{\text{CSR}}) = O(m + n^{1-1/d})$$

The size of ε-sample m is $O(\frac{d+1}{\varepsilon^2} \log \frac{d+1}{\varepsilon})$. because the VC-dimension of stripe range is $d + 1$ [27, 28]. This gives us the following query time:

$$T(\text{EpsRange}_{\text{CSR}}) = O(\max\{n^{1-1/d}, \frac{d}{\varepsilon^2} \log \frac{d}{\varepsilon}\}) = \Omega(n^{1-1/d}).$$

The output is a set $\mathcal{D}_o$ with size $O(\varepsilon n)$, guaranteed to contain $p^{(i)}$ (see Lemma 4). Note that the final time also depends on the choice of parameter ε, which determines the size of output $\mathcal{D}_o$.

Time Complexity for the DAR problem. The exact version also includes the steps iii and iv:

$$T(\text{EpsRange}_{Exact}) = O(m + n^{1-1/d} + |\mathcal{D}_o| \log |\mathcal{D}_o|).$$

Based on Lemma 4, we have $|\mathcal{D}_o| = O(\varepsilon n)$. As a result, by choosing $\varepsilon = n^{-1/3}$, we get the following runtime:

$$T(\text{EpsRange}_{Exact}) = O(\max\{n^{1-1/d}, dn^{2/3} \log n\})$$

Theorem 6. *The EpsRange algorithm solves the relaxed version of DAR problem with linear space usage and $O(\max\{n^{1-1/d}, \frac{d}{\varepsilon^2} \log \frac{d}{\varepsilon}\})$ time. The exact version can also be solved with the same space usage and $O(\max\{n^{1-1/d}, dn^{2/3} \log n\})$ time.*

6.3 Stripe Range Searching with Hierarchical Sampling

As observed in the previous section, the main bottleneck in the runtime of the EpsRange algorithm lies in the range searching subroutines used to solve the SRR problem. These subroutines often suffer from the curse of dimensionality, particularly for general simplices, such as non-orthogonal half-spaces within stripe ranges.

Even algorithms with theoretical sublinear time guarantees tend to perform poorly in practice [37], with a runtime comparable

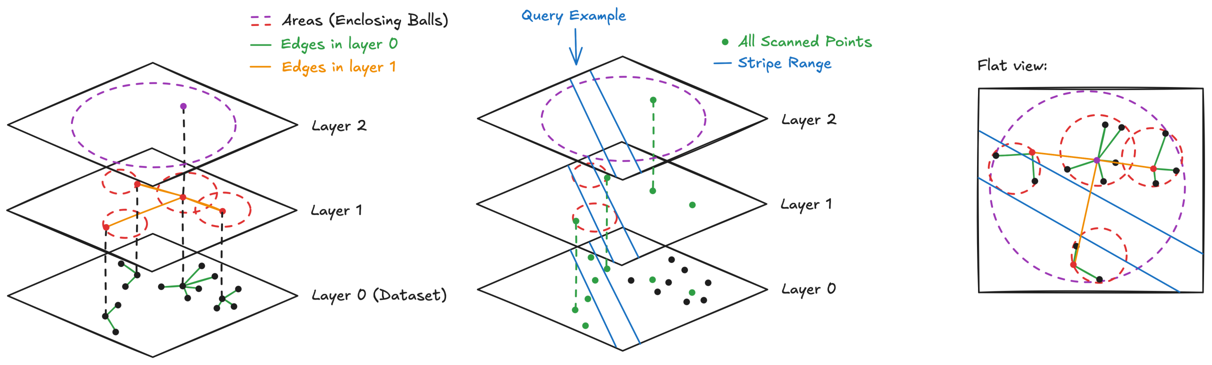

Figure 3: A visual representations of the Hierarchical Sampling structure for solving SRR problem. In this setting, decay rate $r = 4$. (Left) the data structure built during the preprocessing. (Middle) An example of running a query on top of this structure, the green points are all the points scanned during the query phase. (Right) Flat view of the structure and the areas.

to or worse than a naive linear scan over the entire dataset (see experiments in Section 7).

To address this limitation, we propose a practical algorithm for solving the SRR problem, where the ranges are stripes in d dimensions. We empirically show that augmenting EpsRange with this solution leads to significant performance improvements on real-world high-dimensional datasets (called EpsHier algorithm).

We present a hierarchical data structure for solving the stripe range searching problem. We begin with a high-level overview of the structure, followed by a description of the *preprocessing step*, and conclude with the details of the *query phase* of the algorithm.

6.3.1 Overview. We observe that the stripe range queries constructed in the solution of the DAR problem correspond to *narrow regions, especially for small choices of ε*. To exploit this, we construct a hierarchical data structure over the input point set that enables efficient pruning of irrelevant points when answering such range queries.

The core idea is to organize the points in a hierarchy that preserves spatial proximity, inspired by similar structures used in nearest neighbor search [28, 35, 42]. Each point in the hierarchy maintains a list of neighbors, and during preprocessing, we compute and store an *enclosing ball* around each such neighborhood.[7]

During query processing, we employ a simple yet effective heuristic: if the stripe intersects the enclosing ball of a node, we explore the node; otherwise, we safely prune it. This significantly reduces the number of points examined during query time.

Empirical results demonstrate that for the narrow stripe ranges used in the DAR problem, this approach leads to a fast algorithm by eliminating many unnecessary explorations. See Figure 3 (middle), which illustrates the points explored during an example query.

6.3.2 Preprocessing.

Layers. Given the input dataset $\mathcal{D}$, we construct a hierarchical graph $\mathcal{G}(\mathcal{D})$ on top of it. The graph consists of L layers built via recursive random sampling.

We define the base layer as the dataset itself:

$$\mathcal{L}_0 = \mathcal{D}$$

[7]Not necessarily centered at that point.

Algorithm 3 Hierarchical-Sampling-Preprocess$(\mathcal{D}, r)$

Input: The dataset $\mathcal{D}$ and the exponential decay rate r
Output: The hierarchical graph $\mathcal{G}(\mathcal{D})$ which contains a list of layers $\mathcal{L}$, neighboring relations N, and enclosing balls of nodes $\mathcal{B}$.

```
1:  $L \leftarrow \lfloor \log_r n \rfloor$                                   ▷ Number of layers
2:  $\mathcal{L}_\ell \leftarrow []$,  $\forall \ell \le L$                   ▷ Placeholder for the layers
3:  $\mathcal{L}_0 \leftarrow \mathcal{D}$                                    ▷ Base layer
4:  $\mathcal{A}_0(p) = \{p\}$,  $\forall p \in \mathcal{L}_0$                ▷ Area of nodes
5:  $N_0(p) \leftarrow \emptyset$,  $\forall p \in \mathcal{L}_0$             ▷ Neighbors
6:  for layer $\ell = 1$ to $L$ do
7:      $\mathcal{L}_\ell \leftarrow$ Random sample of size $\frac{|\mathcal{L}_{\ell-1}|}{r}$ from $\mathcal{L}_{\ell-1}$
8:      $\mathcal{A}_\ell(p) \leftarrow \emptyset$,  $\forall p \in \mathcal{L}_\ell$
9:      $N_\ell(p) \leftarrow \emptyset$,  $\forall p \in \mathcal{L}_\ell$
10:     for point $p \in \mathcal{L}_{\ell-1}$ do
11:         $c^* \leftarrow \arg\min_{c \in \mathcal{L}_\ell} Dist(p, c)$             ▷ Equation 4
12:         $N_\ell(c^*) \leftarrow N_\ell(c^*) \cup \{p\}$                 ▷ Add neighbor
13:         $\mathcal{A}_\ell(c^*) \leftarrow \mathcal{A}_\ell(c^*) \cup \mathcal{A}_{\ell-1}(p)$   ▷ Update area
14: $\mathcal{B}_\ell(p) \leftarrow$ EnclosedBall$(\mathcal{A}_\ell(p))$,  $\forall \ell \le L, p \in \mathcal{L}_\ell$
15: $\mathcal{L} \leftarrow \{\mathcal{L}_\ell \mid \ell \le L\}$
16: $N \leftarrow \{N_\ell(p) \mid \forall \ell \le L, p \in \mathcal{L}_\ell\}$
17: $\mathcal{B} \leftarrow \{\mathcal{B}_\ell(p) \mid \forall \ell \le L, p \in \mathcal{L}_\ell\}$
18: return $(\mathcal{L}, N, \mathcal{B})$                  ▷ layers, edges, and enclosing balls
```

Each subsequent layer $\mathcal{L}_\ell$ is formed by randomly sampling $\frac{|\mathcal{L}_{\ell-1}|}{r}$ points from the previous layer $\mathcal{L}_{\ell-1}$, where r is a hyperparameter known as the *exponential decay rate*.

Thus, the size of each layer satisfies:

$$|\mathcal{L}_\ell| = \frac{|\mathcal{L}_{\ell-1}|}{r}, \quad \forall\, 1 \le \ell \le L$$

Edges. Once the layer $\mathcal{L}_\ell$ is sampled, each point in $\mathcal{L}_\ell$ serves as a centroid for partitioning the points in the layer below, $\mathcal{L}_{\ell-1}$. Specifically, each point $p \in \mathcal{L}_{\ell-1}$ is connected to its nearest neighbor in $\mathcal{L}_\ell$, forming a *directed edge* from the centroid c^* to p, where

$$c^* = \arg\min_{c \in \mathcal{L}_\ell} ||p - c||_2. \quad (4)$$

This procedure creates a layered structure in which each level induces a graph by connecting lower-layer points to their nearest centroids in the current layer (see Figure 3, left, where each level illustrates the induced graph).

For every node p in the graph $\mathcal{G}(\mathcal{D})$, let $N_\ell(p)$ denote the set of its neighbors at layer $\mathcal{L}_\ell$. For example, in Figure 3 (left), the root node has three red neighbors at layer 1 and five black neighbors at layer 0.

Area of Nodes. For each node p in layer ℓ of the graph $\mathcal{G}(\mathcal{D})$, we associate a set called the *area* of the node, denoted by $\mathcal{A}_\ell(p)$. The area captures the set of points from the base layer that are hierarchically covered by this node.

The definition is recursive. For the base layer, we set:

$$\mathcal{A}_0(p) = \{p\}.$$

For any node p in layer $\ell > 0$, the area is defined as:

$$\mathcal{A}_\ell(p) = \bigcup_{q \in N_{\ell-1}(p)} \mathcal{A}_{\ell-1}(q),$$

where $N_{\ell-1}(p)$ denotes the neighbors (i.e., children) of p in layer $\ell - 1$.

In other words, the area of a node at layer ℓ is the union of the areas of all its children in the layer below.

As illustrated in Figure 3, we represent the area of each node using the smallest enclosing circle, shown with dotted lines.

Enclosing Balls. For each node p at layer ℓ, we define $\mathcal{B}_\ell(p)$ as the *smallest enclosing ball* that covers the area $\mathcal{A}_\ell(p)$. These balls are later used during query time to enable efficient pruning of nodes when searching within a stripe range:

$$\mathcal{B}_\ell(p) = \text{Enclosed-Ball}(\mathcal{A}_\ell(p)).$$

Construction Algorithm. To construct the hierarchical graph $\mathcal{G}(\mathcal{D})$, we employ a bottom-up recursive approach. Starting from the base layer, we iteratively sample a subset of points to form the next layer. At each level, we connect nodes in the current layer to their nearest centroids in the layer above, thereby forming directed edges. After establishing the connections, we compute and update the area sets and corresponding enclosing balls for the newly constructed layer. The full preprocessing procedure is detailed in Algorithm 3.

Analysis. The total space usage for storing the hierarchical layers is linear to the input, since the size of each layer decreases exponentially:

$$\sum_{\ell \le L} |\mathcal{L}_\ell| = \sum_{\ell \le L} \frac{n}{r^\ell} = O(n),$$

assuming $r = O(1)$.

The runtime of Algorithm 3 is dominated by the **for** loop at line 6, where the layers are constructed, and edges and area sets are updated. At each layer ℓ, we sample $|\mathcal{L}_\ell|$ points. Then, for each point in $\mathcal{L}_{\ell-1}$, we compute its nearest neighbor in $\mathcal{L}_\ell$. The time complexity at the layer ℓ is:

$$T_\ell = O\left(d \cdot |\mathcal{L}_\ell| \cdot |\mathcal{L}_{\ell-1}|\right).$$

Summing over all layers, the total runtime becomes:

Algorithm 4 HIERARCHICAL-SAMPLING-QUERY($\mathcal{G}(\mathcal{D}), \mathcal{S}_{f,\ell,u}$)

```
Input: The preprocessed graph $\mathcal{G}(\mathcal{D})$ and the query stripe range $\mathcal{S}_{f,\ell,u}$.
Output: The set of points $\mathcal{D}_o = \mathcal{S}_{f,\ell,u} \cap \mathcal{D}$.
1:  $\mathcal{D}_o \leftarrow \emptyset$                                ▷ Placeholder for result
2:  $candidates \leftarrow \mathcal{L}_L$
3:  for $curr\_layer$ from $L$ to 1 do                              ▷ top-down
4:      $tmp \leftarrow \emptyset$                        ▷ Placeholder for new candidates
5:      for each point $p \in Candidates$ do
6:          if $\mathcal{B}_{curr\_layer}(p) \cap \mathcal{S}_{f,\ell,u} \neq \emptyset$ then
7:              $tmp \leftarrow tmp \cup N_{curr\_layer}(p)$          ▷ Explore the area
8:      $candidates \leftarrow tmp$
9:  for $p \in candidates$ do
10:     if $p \in \mathcal{S}_{f,\ell,u}$ then
11:         $\mathcal{D}_o \leftarrow \mathcal{D}_o \cup \{p\}$
12: return $\mathcal{D}_o$
```

$$\sum_\ell T_\ell = \sum_\ell d \cdot \frac{n}{r^\ell} \cdot \frac{n}{r^{\ell-1}} = \sum_\ell \frac{dn^2}{r^{2\ell-1}} = O(dn^2),$$

again assuming $r = O(1)$.

For computing the smallest enclosing ball of each area, we use Welzl's algorithm, which runs in linear time with respect to the number of points [51].

THEOREM 7. *The preprocessing phase of the hierarchical sampling algorithm takes $O(dn^2)$ time and uses linear space.*

6.3.3 *Query Time.*

Given a stripe range $\mathcal{S}_{f,\ell,u}$, the goal is to report all points in the intersection $\mathcal{S}_{f,\ell,u} \cap \mathcal{D}$. The query process begins at the top of the hierarchy, starting from layer $\mathcal{L}_L$.

At each layer ℓ, we examine whether the enclosing ball $\mathcal{B}_\ell(p)$ of each point $p \in \mathcal{L}_\ell$ intersects the stripe $\mathcal{S}_{f,\ell,u}$. If there is no intersection, the corresponding node is pruned from further exploration. Otherwise, we proceed to explore its neighbors in the next lower layer, denoted by $N_{\ell-1}(p)$.

The full query procedure is provided in Algorithm 4. A visual illustration of this process is shown in Figure 3 (middle).

7 EXPERIMENTS

7.1 Experimental Setup

We conducted our experiments on a server running Ubuntu 20.04, equipped with 123 GB of RAM and a 64-core processor. Our code and other artifacts are available on this GitHub repository.

7.1.1 *Datasets.*

We use both synthetic and real-world datasets for our evaluation. The synthetic datasets are generated using the Zipfian distribution, with variations in dimensionality, dataset size, and distribution parameters. The Zipfian distribution is particularly challenging for range searching due to its inherent skewness and is commonly observed in many real-world scenarios. It allows us to evaluate the algorithms under non-uniform workloads.

For our real-world experiments, we use two datasets: the US Used Cars dataset [40] and the FIFA 2023 dataset [33]. The US Used Cars dataset contains approximately 3 million entries with 66 attributes, while the FIFA 2023 dataset comprises around 300,000 records with 54 attributes. The task in both datasets is to retrieve the items (cars or FIFA teams) that fall within a specified ranking window, as defined by the DAR problem.

7.1.2 Baselines and Methods. We evaluate our methods on all the SRR, DAR, and CSR problems.[8] For the SRR problem, we compare against classical and widely used range searching indices, including KD-Tree [5], R-Tree [26], and Partition Tree [36]. Our proposed method, referred to as Hierarchical Sampling (abbreviated as Hierarchical), is included in the reports. As a baseline, we also include a simple exhaustive search approach, denoted as Exhaustive. To apply KD-Tree and R-Tree structures to stripe queries, we traverse the tree top-down, pruning subtrees whose corresponding hyperrectangles do not intersect the query stripe (similar to Algorithm 4).

For the DAR and CSR problems, we include the Threshold Algorithm (TA) and Fagin's algorithm [25] as baseline methods. Additionally, we consider a simple exhaustive baseline, referred to as Exhaustive, which computes the score for all points, sorts them according to the scoring function, and returns the i-th ranked element. Our proposed algorithms are EpsRange and EpsHier, and are used for the CSR problem.[9] We also include KthLevel, the algorithm based on arrangement construction for our low-dimensional experiments.

We begin by presenting experiments on the intermediate SRR problem to evaluate the core range searching component (section 7.2). We then proceed to the end-to-end evaluation of the full DAR problem (section 7.3).

7.2 Stripe Range Retrieval

7.2.1 Synthetic Dataset. We begin by evaluating the performance of the algorithms on synthetic datasets.

Effect of Stripe Width. For these experiments, the scoring function f is sampled uniformly from the unit hypersphere in $\mathbb{R}^d$. We vary the stripe $\mathcal{S}_{f,\ell,u}$ by adjusting its width, i.e., the number of points it contains. Figure 4 presents the query time performance across different dimensionalities d as a function of stripe width.

We observe that both KD-Tree and R-Tree methods fail to scale effectively to high-dimensional settings, and thus we exclude them from experiments with $d \geq 32$. The proposed Hierarchical Sampling method achieves **up to a 16× speedup** over Exhaustive search, particularly in narrow stripe queries. Such narrow stripes frequently arise when solving the DAR and CSR problems, indicating that this method is also helpful for DAR problem. While the Partition Tree also shows a fast query time in low-dimensional spaces, its performance degrades with increasing dimensionality.

Effect of Dataset Size. Figure 5 illustrates the impact of increasing the dataset size n on the query time performance of the algorithms, for both low- and high-dimensional settings. As observed, the Hierarchical Sampling method consistently outperforms the baselines with a significant speedup.

Preprocessing (Indexing) Phase. All methods evaluated involve a preprocessing phase in which an index structure is constructed. Figure 6 (left) shows the memory usage of these indices as a function of the dataset size n, while the right plot illustrates the corresponding preprocessing time. As shown, the space overhead of Hierarchical Sampling is comparable to that of KD-Tree and R-Tree, both of which are commonly used in existing systems. Moreover, the space usage grows linearly with the dataset size n. The time required to build the Hierarchical Sampling index is also on par with that of the Partition Tree and R-Tree.

7.2.2 Real Datasets. We evaluate SRR queries on real-world datasets, including the US Used Cars dataset as an example of a large-scale dataset, and the FIFA 2023 dataset. The results are presented in Figure 7. As shown, Hierarchical Sampling significantly outperforms both Exhaustive search and the Partition Tree method. In this experiment, multiple random scoring functions are sampled uniformly from the unit hypersphere.

Accuracy of the Output. In all the settings, the output is guaranteed to have a recall of 100%, reporting all the points in the range.

7.3 Direct-Access and Conformal Set Retrieval

7.3.1 Synthetic Dataset.

Effect of Dimension and value of i. Figure 8 (left) presents the query time of various algorithms as a function of data dimensionality. As the dimensionality increases, both Fagin and TA algorithms exhibit poor scalability for the DAR queries. Between the proposed methods, EpsHier consistently outperforms EpsRange, particularly when the dimensionality exceeds 8. The performance degradation of EpsRange in higher dimensions is because of its reliance on Partition Tree-based range searching [37], which suffers from the curse of dimensionality.

Figure 8 (right) illustrates the impact of varying the value of i in the DAR query on the runtime performance. The TA algorithm shows increasing query time as i grows, since it must retrieve the top-i items. In contrast, the runtime of EpsRange remains relatively stable, as it depends primarily on the position of the stripe rather than the value of i. The EpsHier method also shows a moderate increase in runtime for larger i, which can be because of the relative location of the stripes with respect to the dataset. The marginal stripes (smaller values of i) intersect with fewer balls in the hierarchical structure proposed in Section 6.3, and the pruning is more effective in this case. In contrast, stripes that are relatively in the middle require more balls to be explored, because they intersect with more balls, and as a result, the pruning is less effective.

Effect of Dataset Size and ε. Figure 9 (left) shows the effect of dataset size on the query time of the algorithms. We observe significant speedups achieved by both EpsRange and EpsHier compared to the baseline methods.

Figure 9 (right) illustrates the impact of varying the parameter ε. As expected, this parameter influences the size of the output set in the CSR problem. Increasing the value of ε results in a larger output set, whereas smaller values of ε require searching over a larger ε-sample during the initial phase of the algorithm. Both extremes lead to increased query times. The most efficient performance is

[8]We may use DAR and CSR interchangeably in this section.
[9]Whenever not mentioned in the experiment details, the value of ε is chosen such that the conformal set contains less than 20 points.

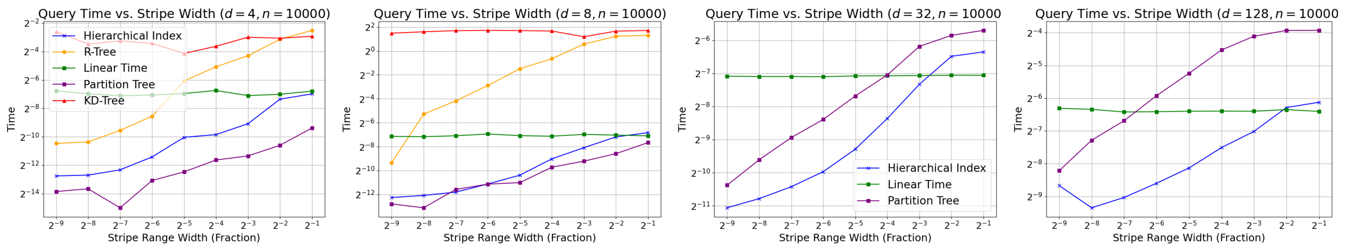

Figure 4: Comparison of SRR query time with respect to the stripe width across different dimensionalities on the synthetic data.

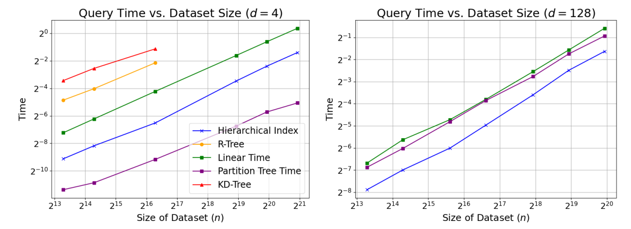

Figure 5: Comparison of SRR query time with respect to dataset size. The higher-dimensional and larger cases do not contain KD-tree and R-tree as methods do not scale well.

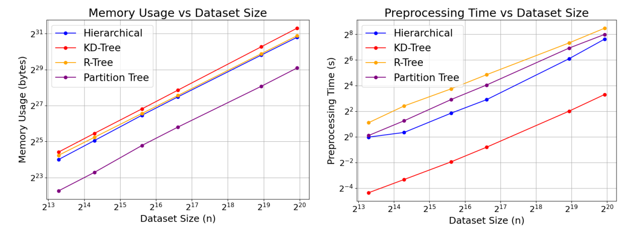

Figure 6: Comparison of SRR indexing time and size with respect to the dataset size for $d = 8$.

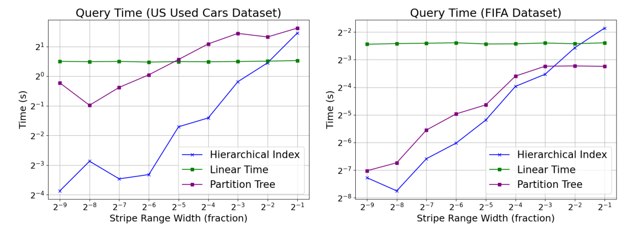

Figure 7: Comparison of SRR query time with respect to the stripe range width. The scoring functions are sampled uniformly at random from a hypersphere.

typically achieved for a moderate value of ε, which may serve as a reasonable default when the user does not specify a preference.

KthLevel Algorithm. We conducted experiments to evaluate the performance of the KthLevel algorithm on smaller datasets in 2D. As shown in Figure 10, the KthLevel algorithm achieves substantial speedups. However, this improvement comes at the cost of increased

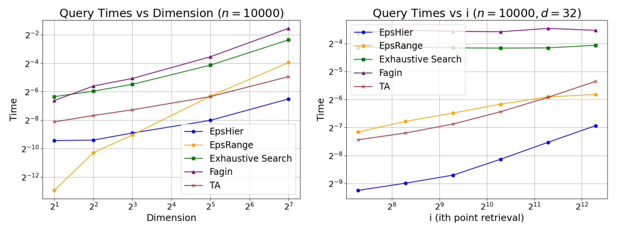

Figure 8: Comparison of CSR query time with respect to the dimension d and the rank i. Here, $\varepsilon = \frac{1}{16}$ for EpsRange and EpsHier.

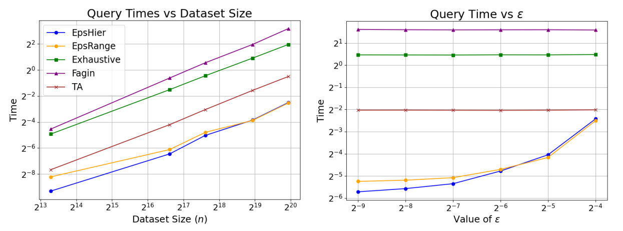

Figure 9: Comparison of query time of CSR with respect to dataset size and ε. In the right plot, $n = 10000$ and the size of returned conformal set is $\varepsilon \cdot n$.

index size, as it stores the k-th level of the arrangement for all values of k.

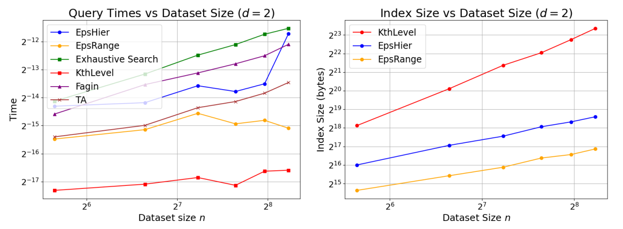

Figure 10: Result of applying KthLevel for solving DAR problem in 2D. Generally, KthLevel does not scale well with dimension d and size n. Here, EpsRange in 2D is equivalent to Eps2D algorithm.

Real Datasets. Figure 12 presents the results of running the algorithms on the US Used Cars dataset, evaluating performance across different values of ε (left) and different values of i (right). Figure 11 shows the corresponding results on the FIFA 2023 dataset.

In both cases, we observe significant speedups achieved by EpsHier, followed by EpsRange, showing their effectiveness in real-world scenarios.

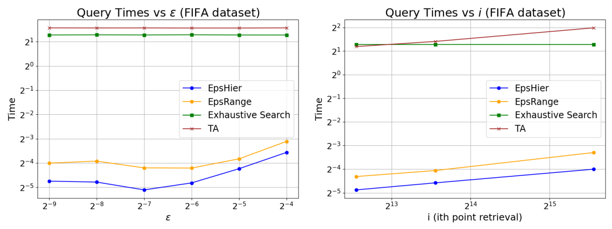

Figure 11: Comparing the CSR query time with respect to value of ε and rank i on FIFA 2023 dataset.

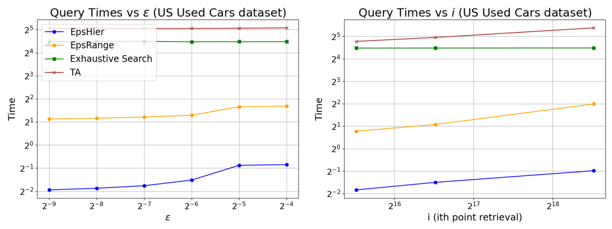

Figure 12: Comparing the query time with respect to value of ε and rank i on US Used Cars Dataset.

Accuracy of the Output. In all the experiments, the returned set is guaranteed to have the $p^{(i)}$, so the recall is always 100%.

8 RELATED WORK

In this section, we briefly review the relevant literature, including prior work on top-k retrieval, direct access queries in databases, and range searching.

Top-k Retrieval. Top-k query processing has been extensively studied across various settings. The foundational Threshold Algorithm (TA) [25] combines sorted and random access to efficiently identify top-k results and has inspired numerous extensions, including cost-based optimizations and early stopping strategies [8]. Probabilistic top-k methods [48] adapt this problem to uncertain data, ranking results based on expected scores or statistical confidence. Index-based approaches such as the onion layer [10] and view-based techniques [18, 29] have been proposed to accelerate queries through structural or materialized reuse. In distributed environments [39], algorithms aim to minimize communication while preserving correctness, and in streaming settings, continuous top-k monitoring [41] uses incremental updates to maintain results in real time. For comprehensive surveys, see [17, 31].

In top-k retrieval, k is a small constant, while in our setting, we aim to find the tuple at rank position i, where i is in $O(n)$. As a result, the algorithms proposed for top-k are inefficient for our setting, specifically when i is not a small constant.

Direct Access Queries in Databases. Recent work in database theory has explored the notion of *direct access* to query answers, aiming to support efficient access to the i-th ranked tuple without materializing the full result [9, 23, 49]. One line of research introduces ranked enumeration for database queries, where tuples can be accessed according to a predefined ranking attribute with provable delay guarantees [49]. Other work extends this to conjunctive queries with aggregation, designing algorithms that enable access to top-ranked answers without fully evaluating the query [23]. Further studies investigate structural conditions, such as bounded treewidth, under which ranked direct access remains tractable [9].

Our paper has a different objective. Specifically, rather than assuming a ranking attribute, *we consider a ranking function as part of the user query*. As a result, the proposed algorithms for direct-access queries cannot be adapted to our setting.

Range Searching. Range searching, specifically simplex range searching, is closely related to our work, particularly in solving the intermediate SRR problem. Some approaches achieve logarithmic query time at the cost of exponential space [3, 21], while others aim for linear space and sublinear query time [14, 28, 36, 37]. Specialized variants have also been studied, including half-space range searching [14] and axis-aligned queries [13]. For a comprehensive overview, we refer the reader to the survey in [1].

To address the SRR problem in practice, we develop a hierarchical sampling approach inspired by data structures commonly used for approximate nearest neighbor search [35]. Particularly, motivated as an intermediate problem for solving direct-access ranked retrieval problem, *our solutions are tailored for narrow ranges*. While our methods do not offer formal theoretical guarantees, they performed efficiently in practical settings. Another indexing technique related to our proposed approach for the SRR problem is the Ball Tree structure [42]. This method constructs a hierarchy of balls over the dataset, where the root node encloses all data points and the points are recursively partitioned into smaller subsets down to the leaves. The key difference between Ball Trees and our hierarchical sampling approach lies in the construction and structure. In our method, each layer is generated by randomly sampling from the previous one, and each layer serves as a set of centroids for the preceding layer. In contrast, Ball Trees are typically binary trees built in a top-down manner. Moreover, our method is not restricted to spherical regions; any convex shape can be used to enclose the nodes, offering greater flexibility in the structure.

9 CONCLUSION

In this paper, we introduced and studied the problem of Direct Access Ranked Retrieval (DAR) and its relaxed variant, Conformal Set Ranked Retrieval (CSR). We proposed an algorithm based on geometric arrangements with logarithmic query time but an exponential space complexity to the number of dimensions, and space-efficient algorithms based on ε-samples. Specifically, providing a hierarchical sampling-based data structure for efficient stripe-range retrieval, we developed an efficient algorithm for the CSR problem that scales to hundreds of dimensions.

Our focus in this paper was on static settings, where the dataset does not change. Extending the scope of our problem to dynamic settings with insert, delete, and update operations is an important next step for future work. Moreover, it would be interesting to study the problem under non-linear ranking functions.

REFERENCES

[1] Pankaj K Agarwal. 2017. Range searching. In *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 1057–1092.

[2] Pankaj K Agarwal, Mark De Berg, Jiri Matousek, and Otfried Schwarzkopf. 1998. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM journal on computing* 27, 3 (1998), 654–667.

[3] Alok Aggarwal, Mark Hansen, and Thomas Leighton. 1990. Solving query-retrieval problems by compacting Voronoi diagrams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 331–340.

[4] Anastasios N Angelopoulos and Stephen Bates. 2021. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511* (2021).

[5] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[6] Nikos Bikakis, Panos K Chrysanthis, Guoliang Li, George Papastefanatos, and Lingyun Yu. 2025. Visual Analytics Challenges and Trends in the Age of AI: The BigVis Community Perspective. *ACM SIGMOD Record* 54, 2 (2025), 66–69.

[7] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, Robert Endre Tarjan, et al. 1973. Time bounds for selection. *J. Comput. Syst. Sci.* 7, 4 (1973), 448–461.

[8] N. Bruno, L. Gravano, and A. Marian. 2002. Evaluating top-k queries over Web-accessible databases. In *IEEE ICDE*. 369–380.

[9] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. 2023. Tractable orders for direct access to ranked answers of conjunctive queries. *ACM Transactions on Database Systems* 48, 1 (2023), 1–45.

[10] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R. Smith. 2000. The onion technique: indexing for linear optimization queries. *ACM SIGMOD* 29, 2 (2000), 391–402.

[11] Bernard Chazelle. 1989. Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society* 2, 4 (1989), 637–666.

[12] Bernard Chazelle. 2000. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press.

[13] Bernard Chazelle and Leonidas J Guibas. 1986. Fractional cascading: I. A data structuring technique. *Algorithmica* 1, 1 (1986), 133–162.

[14] Bernard Chazelle, Leo J Guibas, and Der-Tsai Lee. 1985. The power of geometric duality. *BIT Numerical Mathematics* 25, 1 (1985), 76–90.

[15] Fanchao Chen, Dixin Tang, Haotian Li, and Aditya G Parameswaran. 2023. Visualizing Spreadsheet Formula Graphs Compactly. *Proceedings of the VLDB Endowment* 16, 12 (2023), 4030–4033.

[16] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[17] Gautam Das. 2009. Top-k Algorithms and Applications. In *DASFAA*. 789–792.

[18] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. 2006. Answering top-k queries using views. In *VLDB*. 451–462.

[19] Mohsen Dehghankar and Abolfazl Asudeh. 2025. HENN: A Hierarchical Epsilon Net Navigation Graph for Approximate Nearest Neighbor Search. *CoRR, abs/2505.17368* (2025).

[20] Tamal K Dey. 1998. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry* 19, 3 (1998), 373–382.

[21] David Dobkin, John Hershberger, David Kirkpatrick, and Subhash Suri. 1990. Implicitly searching convolutions and computing depth of collision. In *International Symposium on Algorithms*. Springer, 165–180.

[22] Herbert Edelsbrunner. 1987. *Algorithms in combinatorial geometry*. Vol. 10. Springer Science & Business Media.

[23] Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. 2023. Direct access for answers to conjunctive queries with aggregation. *arXiv preprint arXiv:2303.05327* (2023).

[24] Hazel Everett, Jean-Marc Robert, and Marc van Kreveld. 1993. An optimal algorithm for the ($\leq$ k)-levels, with applications to separation and transversal problems. In *Proceedings of the ninth annual symposium on Computational geometry*. 38–46.

[25] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *ACM PODS*. 102–113.

[26] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[27] Sariel Har-Peled. 2011. *Geometric approximation algorithms*. Number 173. American Mathematical Soc.

[28] David Haussler and Emo Welzl. 1986. Epsilon-nets and simplex range queries. In *Proceedings of the second annual symposium on Computational geometry*. 61–71.

[29] Vagelis Hristidis and Yannis Papakonstantinou. 2004. Algorithms and applications for answering ranked queries using ranked views. *The VLDB Journal* 13, 1 (2004), 49–70.

[30] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)* 40, 4 (2008), 1–58.

[31] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *Comput. Surveys* 40, 4, Article 11 (2008).

[32] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks–a publishing format for reproducible computational workflows. In *Positioning and power in academic publishing: Players, agents and agendas*. IOS press, 87–90.

[33] Stefano Leone. 2022. FIFA 23 Complete Player Dataset. https://www.kaggle.com/datasets/stefanoleone992/fifa-23-complete-player-dataset. Accessed: 2025-07-31.

[34] Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems* 37 (2024), 94871–94908.

[35] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[36] Jiří Matoušek. 1991. Efficient partition trees. In *Proceedings of the seventh annual symposium on Computational geometry*. 1–9.

[37] Jiri Matousek. 1992. Reporting points in halfspaces. *Computational Geometry* 2, 3 (1992), 169–186.

[38] Stefan Meiser. 1993. Point location in arrangements of hyperplanes. *Information and Computation* 106, 2 (1993), 286–303.

[39] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. 2005. KLEE: a framework for distributed top-k query algorithms. In *VLDB*. 637–648.

[40] Ananay Mital. 2020. US Used Cars Dataset. https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset. Accessed: 2025-07-31.

[41] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. 2006. Continuous monitoring of top-k queries over sliding windows. In *ACM SIGMOD*. 635–646.

[42] Stephen M Omohundro. 1989. Five balltree construction algorithms. (1989).

[43] Sajjadur Rahman, Kelly Mack, Mangesh Bendre, Ruilin Zhang, Karrie Karahalios, and Aditya Parameswaran. 2020. Benchmarking spreadsheet systems. In *Proceedings of the 2020 acm sigmod international conference on management of data*. 1589–1599.

[44] Carl M Rebman Jr, Queen E Booker, Hayden Wimmer, Steve Levkoff, Mark McMurtrey, and Loreen Marie Powell. 2023. An Industry Survey of Analytics Spreadsheet Tools Adoption: Microsoft Excel vs Google Sheets. *Information Systems Education Journal* 21, 5 (2023), 29–42.

[45] Glenn Shafer and Vladimir Vovk. 2008. A tutorial on conformal prediction. *Journal of Machine Learning Research* 9, 3 (2008).

[46] Shreya Shankar, Stephen Macke, Sarah Chasins, Andrew Head, and Aditya Parameswaran. 2022. Bolt-on, compact, and rapid program slicing for notebooks. *Proceedings of the VLDB Endowment* 15, 13 (2022), 4038–4047.

[47] Micha Sharir, Shakhar Smorodinsky, and Gábor Tardos. 2000. An improved bound for k-sets in three dimensions. In *Proceedings of the sixteenth annual symposium on Computational geometry*. 43–49.

[48] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. 2004. Top-k query evaluation with probabilistic guarantees. In *VLDB*. 648–659.

[49] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2024. Ranked enumeration for database queries. *ACM SIGMOD Record* 53, 3 (2024), 6–19.

[50] Vladimir N Vapnik and A Ya Chervonenkis. 2015. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity: festschrift for alexey chervonenkis*. Springer, 11–30.

[51] Emo Welzl. 2005. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings*. Springer, 359–370.

[52] Jinjin Zhao, Avigdor Gal, and Sanjay Krishnan. 2023. Data Makes Better Data Scientists. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–3.

APPENDIX

A LOWER BOUNDS AND OPTIMALITY OF OUR ALGORITHMS

In this section, we analyze the complexity of the DAR problem and show the near-optimality of our proposed algorithms. Specifically, the well-studied *half-space range counting* problem [1, 11] reduces to our problem. Hence, the complexity of the state-of-the-art algorithms for half-space range counting provides a lower bound for the DAR problem.

The half-space range counting problem is defined as follows: given a point set in $\mathbb{R}^d$ and a query half-space H, determine the number of points lying within H.

Lemma 8. *Let $\mathcal{A}$ be an algorithm that solves the DAR problem with query time $T(\mathcal{A})$ and space complexity $S(\mathcal{A})$. Then, using $\mathcal{A}$, one can solve an instance of the half-space range counting problem using space $S(\mathcal{A})$ and query time:*

$$\Theta(\log n \cdot T(\mathcal{A})).$$

□

Proof. To count the number of points within a given half-space H, we can perform a binary search over $k \in \{1, 2, \dots, n\}$. In each iteration, we query $\mathcal{A}$ to retrieve the k-th ranked point $p^{(k)}$. If $p^{(k)} \in H$, then the count of points in H is at least k; otherwise, it is less. This process requires $\Theta(\log n)$ queries to $\mathcal{A}$, resulting in total query time $\Theta(\log n \cdot T(\mathcal{A}))$. □

It is known that the query time for half-space range counting in $\mathbb{R}^d$ with m units of space satisfies the following lower bound [11]:

$$\Omega\left(\frac{n}{m^{1/d}\log n}\right). \tag{5}$$

Applying Lemma 8, this implies a lower bound for the DAR problem:

$$T(\mathcal{A}) \cdot \log n \geq \frac{n}{m^{1/d}\log n} \quad \Longrightarrow \quad T(\mathcal{A}) \geq \frac{n}{m^{1/d}\log^2 n}.$$

Now, these are the two important observations:

Observation 1. If the algorithm uses linear space, i.e., $m = O(n)$, then the query time cannot be faster than $O(n^{1-\frac{1}{d}})$ up to logarithmic factors.

Observation 2. If the algorithm uses exponential space, i.e., $m = 2^{\Theta(d)}$, then the lower bound becomes $\Omega(\log n)$. In fact, our exact algorithm KthLevel has logarithmic query time but requires exponential space, which matches this bound. Note that querying KthLevel n times is equivalent to sorting all n elements, which takes $\Omega(n \log n)$ time.

Conclusion. These results demonstrate that our algorithms for the DAR problem are optimal up to logarithmic factors, given the space-time trade-offs imposed by known lower bounds.